Volume Graphics (2007)
H. - C. Hege, R. Machiraju (Editors)

Surface Reconstruction from Noisy Point Clouds using Coulomb Potentials

A. C. Jalba and J. B. T. M. Roerdink

Institute for Mathematics and Computing Science,
University of Groningen, The Netherlands

Abstract

We show that surface reconstruction from point clouds without orientation information can be formulated as a convection problem in a force field based on Coulomb potentials. To efficiently evaluate Coulomb potentials on the volumetric grid on which the evolving surface (current approximation to the final surface) is convected we use the so called 'Particle-Particle Particle-Mesh' (PPPM) algorithm from molecular dynamics, fully implemented on modern, programmable graphics hardware. Our approach offers a number of advantages. Unlike distance-based methods which are sensitive to noise, the proposed method is highly resilient to shot noise since global Coulomb potentials are used to disregard outliers due to noise. Unlike local fitting, the long-range nature of Coulomb potentials implies that all data points are considered at once, so that global information is used in the fitting process. The method compares favorably with respect to previous approaches in terms of speed and flexibility and is highly resilient to noise.

Categories and Subject Descriptors (according to ACM CCS): G.1.2 [Numerical Analysis]: Approximation - Approximation of surfaces and contours; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Physically based modeling; I.4.10 [Image Processing and Computer Vision]: Image Representation - Volumetric;

1. Introduction

Reconstructing 3D surfaces from point clouds allows one to obtain digital representations of physical objects for rendering purposes. This is a challenging problem because the real surface has unknown topology, surface orientation at the sample points is unknown, acquired data are often non-uniform and contaminated by noise, and reconstruction can be computationally very expensive.

Here we present a novel, physically-motivated method for surface reconstruction without orientation information. We use a fast convection algorithm (inspired by the tagging method of Zhao *et al.* [ZOF01]) to attract the evolving surface towards the data points. The force field in which the surface evolves is based on Coulomb potentials evaluated on a volumetric grid using the fast 'Particle-Particle Particle-Mesh' (PPPM) algorithm [HE88] from molecular dynamics. Due to these potentials, the evolving surface is attracted towards the data points and outliers due to noise are removed. The final implicit surface is polygonized using Bloomenthal's polygonizer [Blo94].

Our formulation offers a number of advantages. Implicit surface fitting methods based on signed distance functions are very sensitive to outliers (shot noise), since the shortest distance from *all* input data points is computed. In our approach, global Coulomb potentials are used to neglect such outliers. In contrast to local fitting methods, Coulomb potentials represent global (long-range) interactions that take all data points into account. We exploit a very efficient method for computing Coulomb potentials which is implemented on graphics hardware, obtaining a reconstruction algorithm with good scaling properties.

The contributions of this paper include:

- An improved method for surface reconstruction based on an implicit surface representation which only requires information about the position of the samples and is robust to the presence of noise and missing data.
- Algorithms running on Graphics Processing Units (GPUs) which accelerate the computation of Coulomb potentials.

The main advantage of our approach is improved robustness of surface reconstruction with respect to noise. Further-

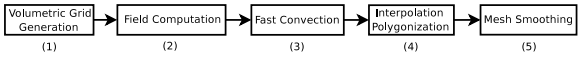

Figure 1: *Algorithmic flow diagram of the proposed method.*

more, minimizing the bending energy of the surface ensures that the final triangulated surface is smoother than piecewise linear. We show that the proposed method quickly reconstructs surfaces of large models and tolerates large amounts of Gaussian and shot noise.

2. Related Work

Many reconstruction methods use implicit surface representations since these can deal with objects of arbitrary topology, perform Boolean operations on surface primitives and fill holes automatically. A common approach is to compute a signed distance function and represent the reconstructed implicit surface by an iso-surface of this function [HDD*92, CL96]. This requires that one can distinguish between the inside and outside of closed surfaces. To get the orientation one may fit a local tangent plane [HDD*92] or use tensor-voting [TM98]. Both methods are sensitive to noise since they require accurate normal estimates. Zhao *et al.* [ZOF01] use the level-set formalism to obtain a method for noise-free surface reconstruction which can handle complicated topology and deformations. It leads to a reconstructed surface which is smoother than piecewise linear, but the method is very sensitive to shot noise.

Other recent approaches are based on Radial Basis Functions or RBFs [CBC*01, KSH04, DTS02]. When using globally supported RBFs [TO02, CBC*01] one has to solve a large and dense linear system of equations which is ill-conditioned [KBH06]. Also, the methods are very sensitive to noise. Compactly-supported RBFs allow local control and reduce computation [OBS03]. RBFs with volumetric regularization can handle noisy and sparse range data sets [DTS02]. The 'partition of the unity implicits' method of Ohtake *et al.* [OBA*03] combines algebraic patches and RBFs. To deal with noisy data, Carr *et al.* [CBM*03] fit a RBF to the data combined with a smoothing during the evaluation of the RBF. Kojekine *et al.* [KSH04] use compactly-supported RBFs and an octree data structure, resulting in a band-diagonal matrix with reduced computational cost. Fleishman *et al.* [FCOS05] proposed an adaptive method based on the Moving Least-Squares algorithm. More recently, Kazhdan proposed a Fourier transform method with standard iso-surfacing [Kaz05], and an improved, geometrically adaptive method [KBH06] with quadratic complexity. The latter regards reconstruction as a Poisson problem tackled using efficient Poisson solvers. However, both methods require orientation information and are noise sensitive.

3. The Proposed Method

The computational flow diagram of our method is shown in Fig. 1. The method starts by assigning the input sample points to grid cells, using cloud-in-cell (CIC) interpolation (first step in Fig. 1). The Coulomb potentials in which the evolving surface $\mathbf{\Gamma}$ (see below) is convected (step 3) are computed using the PPPM method (step 2). Prior to polygonization, the resulting implicit surface is interpolated by employing a reaction-diffusion process [JR06]. Finally, we employ Bloomenthal's polygonizer [Blo94] to turn the implicit surface into a triangulated one (second part of step 4), and use a mass-spring system, enhanced with a bending-energy minimizing term, to obtain a larger degree of smoothness (step 5).

3.1. Field Computation

We denote by S the input set of point samples (points, lines, etc.) which are assumed to lie on or near the surface $\partial \boldsymbol{M}$ of an unknown object M. The problem is to reconstruct accurately the indicator function χ of the object (defined as 1 at points inside the object, and 0 at points outside).

The reconstruction problem is formulated as the convection of a flexible enclosing surface $\mathbf{\Gamma}$ in a conservative velocity field $\boldsymbol{v}$ created by the input data points, described by the differential equation

$$\frac{d\mathbf{\Gamma}}{dt} = -\nabla\phi\left(\mathbf{\Gamma}(t)\right). \tag{1}$$

where ϕ is the potential, i.e., $\boldsymbol{v} = -\nabla\phi$. We regard the input sample points s_i, $s_i \in S$, as electric charges q_i, located at positions $\boldsymbol{r}_i$, $i = 1, 2, \ldots N$, so that ϕ becomes the Coulomb potential, that is, the sum of potentials generated by each charge taken in isolation:

$$\phi(\boldsymbol{r}_i) = \sum_{j \neq i} \frac{q_j}{4\pi\varepsilon_0 |\boldsymbol{r}_i - \boldsymbol{r}_j|}. \tag{2}$$

We use a fast convection algorithm (see Section 3.2) which needs to evaluate Coulomb potentials not only at the positions of the sample points, but at all centers of grid cells. Naive evaluation of the potentials at *all* grid positions is expensive for large grid resolutions, so we need fast adaptive solvers to approximate them.

3.1.1. The PPPM method

An efficient approach for computing Coulomb potentials at *all grid positions*, which also lends itself to a GPU implementation (see Section 4) is the so-called 'Particle-Particle Particle-Mesh' (PPPM) method from molecular dynamics [HE88]. The PPPM method splits the Coulomb potential into a short range direct interaction part (PP) and a contribution from the mesh (PM). Accordingly, the Coulomb energy at position $\boldsymbol{r}_i = [x_i, y_i, z_i]$ is

$$W_i = \frac{1}{2} \sum_{j=1}^{M} W_{ij}^{direct} + W_i^{mesh} \tag{3}$$

The direct part of the Coulomb energy of a pair of charges (q_i, q_j) separated by distance vector $\boldsymbol{r}_{ij}$ is given by the Coulomb energy minus a correction term

$$W_{ij}^{direct}(\boldsymbol{r}_{ij}) = \begin{cases} q_i q_j \left(\frac{1 - W^c(\boldsymbol{r}_{ij})}{\left(|\boldsymbol{r}_{ij}|^2 + \varepsilon^2\right)^{1/2}} \right) & \left|\boldsymbol{r}_{ij}\right| \leq R_c \\ 0 & \left|\boldsymbol{r}_{ij}\right| \geq R_c. \end{cases} \quad (4)$$

Here R_c is the direct interaction cut-off radius and $\varepsilon \ll 1$ is a "softening parameter" with dimension of length, introduced to simplify the treatment of singularities occurring when $\boldsymbol{r}_{ij} = 0$. The direct contribution to the Coulomb potential vanishes at R_c, and for $\left|\boldsymbol{r}_{ij}\right| \geq R_c$, there is only a mesh contribution. The correction term $W^c(\boldsymbol{r})$ compensates for the part of the interaction already covered by the mesh potential (see Eq. (9) below).

The mesh potential ϕ^{mesh} is obtained by solving Poisson's equation on the grid

$$\nabla^2 \phi(\boldsymbol{r}_i) = -\rho(\boldsymbol{r}_i), \quad (5)$$

where $\rho(\boldsymbol{r}_i)$ and $\phi(\boldsymbol{r}_i)$ are the charge density and the electric scalar potential at grid point $\boldsymbol{r}_i$, respectively. Here $\rho(\boldsymbol{r}_i)$ is the charge per grid cell volume, and is computed in two steps. In the first step we use the linear charge assignment scheme in [HE88]. Every charge q_i is distributed over its 6 surrounding grid points, and the charge at grid point $\boldsymbol{r}_i$ is computed as

$$q(\boldsymbol{r}_i) = \sum_{k=1}^{M} q_k H(\boldsymbol{r}_i - \boldsymbol{r}_k). \quad (6)$$

Here $H = H_x H_y H_z$ is the weight of a charge located at $\boldsymbol{r}$,

$$H_x(r_{i,x} - r_x) = \begin{cases} 1 - \frac{|r_{i,x} - r_x|}{h_x} & \left|r_{i,x} - r_x\right| < h_x \\ 0 & \left|r_{i,x} - r_x\right| \geq h_x, \end{cases}$$

where M is the size of the grid, h_x is the mesh grid spacing in dimension x, and H_y, H_z are defined similarly. In the second step, the charges are spread over a larger neighbourhood of grid points, in order to produce a smooth total charge distribution. This step is implemented using the approach in [BLdL98], in which the charges are spread by a diffusion process. The method proceeds by solving Poisson's equation, Eq. (5), on the mesh. Then, the mesh-energy term is computed as a weighted sum over the same grid points used in the first charge-assignment step

$$W_i^{mesh} = q_i \sum_k H(\boldsymbol{r}_i - \boldsymbol{r}_k)\phi(\boldsymbol{r}_k) - W_{Gauss,i}^{self}, \quad (7)$$

where $W_{Gauss,i}^{self}$ is a correction term for the mesh energy which a particle experiences from its own charge distribution (the self-energy). This constant term per particle is given by

$$W_{Gauss,i}^{self} = q_i^2/\sigma, \quad (8)$$

where $\sigma = 2\sqrt{DN_t}$, D is the diffusion coefficient, and N_t is the number of time steps of the diffusion process (N_t and D can be deduced using the cut-off radius parameter R_c, see [BLdL98] for details). For a Gaussian charge distribution, obtained by solving a linear diffusion equation, the correction term $W^c(\boldsymbol{r})$ in Eq. (4) is given by

$$W_{Gauss}^c(\boldsymbol{r}) = \mathrm{erf}\,(\boldsymbol{r}/\sigma), \quad (9)$$

where 'erf' denotes the error function. Note that we have neglected all constants in the description of the PPPM method. Assuming that $q_i = q = 1$, the electric scalar potential is given by $\phi(\boldsymbol{r}_i) = W(\boldsymbol{r}_i)/q_i = W(\boldsymbol{r}_i)$, $i = 1, 2, \ldots, M$.

Setting the softening parameter ε in (4) to some very small value will result in large potentials at input sample positions. This has the desirable effect that regional maxima of the Coulomb potential exist at the locations of the data points, for noise-free data. If the input data set is noisy, however, then a larger value of the parameter ε is required in order to cancel regional maxima located at outlier positions. In practice, we fix the value of this parameter, but for noisy data sets, we increase the size of the neighbourhood of a point in which the direct potential is computed (see Section 6).

3.2. Fast Convection based on Tagging

We use the tagging algorithm of Zhao *et al.* [ZOF01], adapted for the scalar electric potential $\phi(\boldsymbol{r})$, which we briefly summarize here. The algorithm starts by labeling points on the bounding box of the computational domain as exterior and all other points as interior. Then, those interior points that have at least one exterior neighbour are labeled as temporary (unknown) boundary points and are inserted into a *minimum-sorted heap*. Next, the remaining interior points are swept to march the temporary boundary points inwards towards the input points, as follows. The temporary boundary point with the smallest potential (which is on top of the heap) is checked to see whether it has at least one interior neighbour with a smaller or equal potential value. If it does not have such a neighbour, the point is taken out from the heap, turned into an exterior point, and all its interior neighbours are inserted into the heap. Otherwise, the point is removed from the heap, turned into a final boundary point, and none of its neighbours is added to the heap. This process is repeated until no further movement of the temporary boundary points is possible.

Clearly, this algorithm moves the points of the evolving surface uphill, in the direction of the gradient $\nabla\phi$, towards their closest point(s) in the data set. Moreover, each point of the final surface satisfies $\nabla\phi = 0$. Therefore the tagging algorithm can be considered as a fast steady-state solver for similar convection problems.

3.3. Surface Interpolation and Polygonization

Once the classification of the grid points into interior, boundary and exterior has been completed, i.e., the characteristic function χ of the model has been computed, one can use Bloomenthal's method [Blo94] to polygonize the implicit

surface given by the zero level set of the scalar field f defined by

$$f(x,y,z) = \begin{cases} -1 & \text{if (x,y,z) is labeled as INTERIOR} \\ 0 & \text{if (x,y,z) is labeled as BOUNDARY} \\ 1 & \text{if (x,y,z) is labeled as EXTERIOR.} \end{cases} \quad (10)$$

Direct polygonization will cause "staircase" artefacts in the resulting mesh. A better approach is to interpolate the implicit surface using a reaction-diffusion process described by the following PDE

$$\frac{\partial u}{\partial t} = \mu \nabla^2 u + |f|\,(f-u), \quad (11)$$

where μ is a small regularization constant which controls the amount of smoothing, u is the concentration of diffusing material, with the volume f as initial condition, $u(t=0) = f$. The last term in Eq. (11) ensures that the reaction-diffusion equation reaches a steady state not far from the original values of f.

After computing the potentials a smooth scalar field u emerges and by tracing its zero iso-surface, the implicit surface is turned into a triangulated one. Since boundary voxels form thin bands along surface borders, only a small number of iterations is required, resulting in fast computation. The resulting triangulated surface is used as initialization for a more computationally demanding smoothing method based on a mass-spring system. That is, the input mesh is regarded as a mass-spring system having particles with small, equal masses at its vertices connected by springs along its edges. Then, the potential energy of a particle p_i of the mass-spring system due to its interactions with neighbouring particles p_j, $j \in \mathcal{N}_i$ is given by

$$E_i = \sum_{j \in \mathcal{N}_i} \alpha E_{s_{ij}}(\boldsymbol{r}_{ij}) + (1-\alpha) E_{b_{ij}}(\boldsymbol{r}_{ij}, \boldsymbol{n}_i), \quad (12)$$

where the first term represents the energy of the spring connecting the particles, the second term is the *bending energy*, and α is a scalar weight. Further, $\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$ is the vector separating the two particles, and $\boldsymbol{n}_i$ is the normal of the mesh at p_i. For further details we refer to [JR06].

4. Implementation

We now describe GPU implementations of most computationally-demanding parts of the proposed method. We present (i) an iterative method for the computation of the reaction-diffusion equation, Eq. (11), and (ii) an implementation of the PPPM method. The mass-spring system was also implemented on graphics hardware using the layout of [BFGS03] for efficiently storing sparse matrices on the GPU. Referring to Fig. 1, the field computation step and the two smoothing methods based on reaction-diffusion and mass-spring systems are implemented on GPU hardware. The convection and polygonization steps are still performed on the CPU, as current GPUs do not permit an efficient implementation of a sorted heap, nor do they yield better performance than CPUs for polygonization of implicit surfaces.

4.1. Reaction-Diffusion

Eq. (11) is discretized using finite differences (forward differences in time and central differences in space), such that the discrete update rule is

$$u^{n+1}_{i,j,k} \leftarrow u^n_{i,j,k} + \Delta t \left(\mu \sum_{(l,m,n) \in \mathcal{N}_6} u^n_{i+l,j+m,k+n} - 6u^n_{i,j,k} + \left|f_{i,j,k}\right| \left(f_{i,j,k} - u^n_{i,j,k}\right) \right), \quad (13)$$

where Δt is the time step ($\Delta t \leq \frac{\Delta x \Delta y \Delta z}{6\mu}$ for stability reasons), n is the current iteration, and $\mathcal{N}_6$ denotes the neighbours of location (i,j,k) using 6-connectivity. This discrete update rule can be straightforwardly implemented on GPU hardware by making use of a fragment program which computes (13) at each iteration, based on two textures and a ping-pong approach for render-to-texture.

4.2. Coulomb Potentials

4.2.1. Mesh Contribution

The mesh contribution is obtained by evaluating Poisson's equation on the grid (see Section 3.1), after the initial charge distribution has been computed using linear diffusion. Both the diffusion and Poisson equations needed to compute the mesh contribution are evaluated in a fashion similar to that described in the previous subsection.

4.2.2. Direct Part

The direct part of the Coulomb potential needs some extra consideration. At first glance, its computation may seem similar to that of distance transforms, which can be efficiently implemented on graphics hardware using z-buffer depth tests or blending. Nevertheless, in our case the minimum computation needed for distance transforms is replaced by addition, see (3) and (4). Although the summation in (3) can be performed using alpha blending or using the OpenGL accumulation buffer, a major issue is the limited precision available. Therefore, we use a different approach based on the concept of splatting. Accordingly, a fragment program accumulates different contributions due to nearby charges which splat (or distribute) values of the 'splatting kernel' given in (4). The details are as follows.

For a given cut-off radius R_c, we build a collection T_{R_c} (of size $|T_{R_c}| = R_c + 1$) of textures, each texture representing the contribution of a charge q_i in a plane $z = z_i + r$, with $r \in [-R_c; R_c]$ (assuming that the coordinate system has the origin in $\boldsymbol{r}_i = [x_i, y_i, z_i]$). Then, each texture $t_{z,R_c,\varepsilon} \in T_{R_c}$ encodes the influence of the charge in the plane $z = z_i + r$, when

Algorithm 1 The direct part of the Coulomb potential.

```
for each slice k do
  Clear the color buffer of the p-buffer;
  for l:=-R_c to R_c do
    z:=abs(l);
    Map textures t_acc and t_{s,z,R_c,ε}, with s = 0,1,2,3;
    for y:=0 to H-1 do
      for x:=0 to W-1 do
        if any grid location (x+s,y,l+k), s = 0,1,2,3, contains charge then
          Read rectangular region ((x−R_c)/4, y−R_c, (x+R_c)/4+1, y+R_c+1) to texture t_acc;
          Pass the charges at (x+s,y,l+k), s = 0,1,2,3, as parameters to the fragment program;
          Draw quad at ((x−R_c)/4, y−R_c, (x+R_c)/4+1, y+R_c+1);
        x:=x+4;
  Read the color buffer into slice k;
```

x and y vary in the interval $[-R_c, R_c]$, *i.e.*,

$$t_{z,R_c,\varepsilon}(x,y) \leftarrow \frac{1-\operatorname{erf}\left(\frac{d}{\sigma}\right)}{d}, \tag{14}$$

where $d = \sqrt{(x-R_c)^2+(y-R_c)^2+z^2+\varepsilon^2}$. The kernels are packed into 4-component RGBA textures along rows, such that four operations can be performed without the need for extra computations. The input volumetric grid on which the potentials are to be evaluated is packed similarly. This implies that the width of each texture needs to be $2R_c+4$ instead of $2R_c+1$ and one needs to keep four times as many textures as initially in the collection T_{R_c}, a set for each possible shift $s \in \{0,1,2,3\}$ along the x axis, *i.e.*,

$$t_{s,z,R_c,\varepsilon}(x+s,y) \leftarrow t_{z,R_c,\varepsilon}(x,y).$$

The pseudo-code of the algorithm for the direct part of the Coulomb potential is given in Algorithm 1. Note that, in addition to the collection T_{R_c} of RGBA textures, the program also uses a temporary RGBA texture $\mathtt{t}_{acc}$ in which rectangular regions of the p-buffer (storing the values of the direct part of the Coulomb potential, for the current slice `k`) are read whenever the region of influence of any charge located at positions $(x+s,y,l+k)$, $s=0,1,2,3$ intersects the slice plane. This texture, along with textures $\mathtt{t}_{s,z,R_c,\varepsilon}$, $s=0,1,2,3$, is passed as input to a fragment program which (i) performs lookups into textures $\mathtt{t}_{s,z,R_c,\varepsilon}$ to retrieve 4-component vectors $\boldsymbol{t}_0,\boldsymbol{t}_1,\boldsymbol{t}_2,\boldsymbol{t}_3$, (ii) computes the weighted sum $S = q_0\boldsymbol{t}_0 + q_1\boldsymbol{t}_1 + q_2\boldsymbol{t}_2 + q_3\boldsymbol{t}_3$, where q_0,q_1,q_2,q_3 are the charges at locations $(x+s,y,l+k)$, $s=0,1,2,3$, and (iii) sets the color of the fragment to $C = S + \boldsymbol{t}_{acc}$, where $\boldsymbol{t}_{acc}$ is the 4-component vector extracted from texture $\mathtt{t}_{acc}$. When $k+l$ is outside the range given by the number of slices of the volume, we simply skip the computation for the current l.

The complexity of the PPPM method is thus linear with the size of the computational grid. The obtained speed-up ranges from 4 to 8 compared to a CPU implementation.

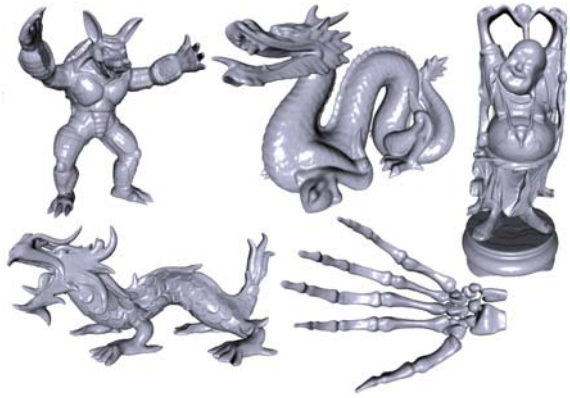

Figure 2: *Reconstruction of large models, see Table 1.*

5. Results

5.1. Large Data Sets

The parameters of the method were set as follows. For the PPPM method, the cut-off radius R_c was set to 8, and we computed an approximate solution to Poisson's equation using $N_p = 30$ iterations of Jacobi's method. Implicit surface interpolation was implemented as discussed in Section 4.1; we set $\mu = 0.05$ and used $N_m = 20$ iterations. The parameters of the Verlet integrator of the mass-spring system were dt $= 0.1$ and $t = 10$. The weight in Eq. (12) was set to $\alpha = 0.1$, to emphasize the bending-energy minimizing term. Further, to facilitate the relaxation of the mesh structure into a smooth configuration, the rest lengths of the springs were set to 95% of the initial edge lengths. Finally, the largest dimension of the computational grid was set to 400.

The meshes resulting from this experiment are shown in Fig. 2. All computations were performed on a system with an Opteron processor and a GeForce FX 7900 GTX GPU. Timings (in seconds) are given in Table 1. The most expensive computations are the convection algorithm and the PPPM method. The time taken to reconstruct nicely either of the models Happy Buddha, Dragon, Hand or Asian Dragon (see Fig. 2) is well under one minute, whereas the Armadillo model needs roughly 70 seconds. The sixth column of Table 1 shows the approximation error – an estimate of the quality of the reconstruction. This error is an upper bound for the average distance from the data points to the surface, and it is computed as the average distance from the data points to the centers of mass of the mesh triangles. The error is given in percentages of the diagonal of the bounding box of the data points.

5.2. Noise Data

5.2.1. Shot Noise

We changed a certain amount of empty voxels by assigning them the value one, *i.e.*, the same numeric value used

Table 1: *Statistics, reconstruction quality and timings.*

Model	Points	Grid	Vertices	Triangles	Error (%)	Field comput. Time (s)	Conv. Time (s)	Interp. Time (s)	Smoothing Time (s)	Total Time (s)
Buddha	543,625	170x400x170	283,964	567,964	**0.07**	12.2	9.6	0.9	2.1	**25.2**
Armadillo	172,974	337x400x307	359,490	718,976	**0.04**	21.5	39.2	3.5	4.2	**69.1**
Dragon	433,375	400x284x184	371,988	743,976	**0.08**	16.4	18.4	1.5	4.1	**40.8**
Hand	327,323	400x283x143	179,484	359,004	**0.04**	9.9	15.7	1.3	1.8	**29.2**
Asian Dragon	3,609,600	400x226x269	199,594	399,188	**0.05**	14.3	23.8	1.8	2.0	**42.7**

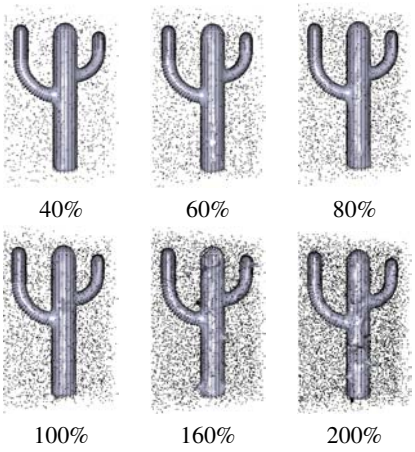

Figure 3: *Shot noise. The number of corrupted voxels is given as a percentage of the number of non-empty grid cells.*

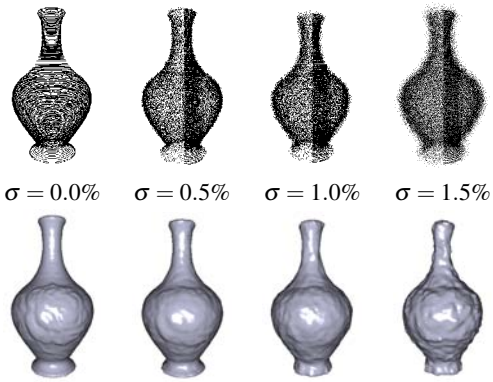

Figure 4: *Gaussian noise with zero mean and deviation σ (a percentage of diagonal size of the bounding box);* first row*: non-empty grid cells,* second row*: reconstructed surfaces.*

to assign the input points. The number of corrupted voxels is expressed as a percentage of the number of non-empty voxels. We used nearest-neighbour interpolation for grid assignment, as this results in a binary volume and represents a fair experimental setting, without a-priori information. The results are shown in Fig. 3; the number of non-empty voxels was 3,337. The cut-off radius R_c of the PPPM method was increased from 8 to 30. A larger cut-off radius results in a larger support of the PP kernel covering most of the exterior volume around the object, which will be correctly labeled as exterior. Note that the method is able to reconstruct the surface of the cactus shown in Fig. 3 even when as much as 100% of the non-empty voxels have been corrupted by noise. Thus, the new method tolerates twice as much shot noise as compared to [JR06].

5.2.2. Gaussian Noise

The input points were perturbed by zero-mean Gaussian noise with standard deviations $\sigma = 0.5, 1.0, 1.5(\%)$, expressed as percentages of the length of the diagonal of the bounding box; see Fig. 4. The grid size was $140 \times 146 \times 250$. The parameters of the method were set as in the previous section, except that the stopping time t of the mass-spring system was increased from 10 to 20. Unlike methods which rely on distance transforms, our method can cope with large amounts of Gaussian noise. For example, in the third case ($\sigma = 1.0\%$) shown in Fig. 4, 1% of the diagonal of the bounding box means that $\sigma = 3.2$, which implies that the coordinates of most points were randomly translated in the interval $[-9.6; 9.6]$. Yet, even in these cases the method is able to output smooth surfaces.

5.3. Comparison to Other Methods

We compare the results of our method to those obtained using Power Crust [ACK01], Multi-level Partition of Unity implicits (MPU) [OBA*03], the method by Hoppe *et al.* [HDD*92], the FFT method in [Kaz05] and the Poisson-based method [KBH06]. The experiment was performed using the Stanford bunny data set consisting of 362,000 samples assembled from ten range images. The normal at each sample was estimated from the positions of the neighbours of the sample (as in ref. [KBH06]), as this is required by the MPU, FFT and Poisson methods. The results are shown in Fig. 6. Since this data set is noisy, interpolating methods such as the Power Crust generate very noisy surfaces with holes due to the non-uniformity of the samples. Hoppe's *et al.* method [HDD*92] generates a smooth surface, but some holes are still visible due to the non-uniform distribution of samples, which the method cannot properly handle. The MPU method yields a smooth surface without holes, but with some artefacts due to the local nature of the fitting which

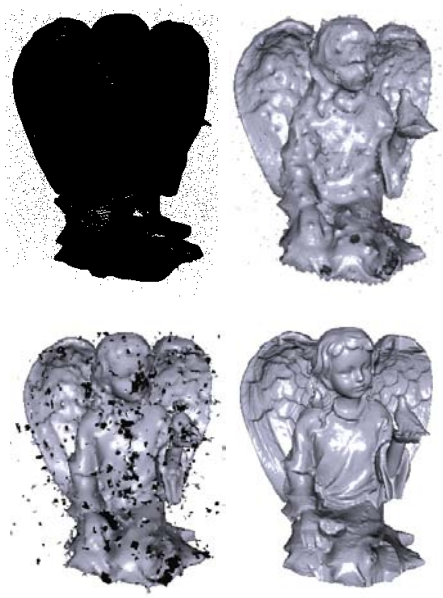

Figure 5: Left-to-right, top-to-bottom*: noisy range data set with* 2,008,414 *input points and* 4000 *outliers; reconstructed surfaces by (i) the level-set method of Zhao* et al. *[ZOF01] (*2,139 *sec.), (ii) Hoppe* et al. *[HDD*92] (*610 *sec.), and (iii) the proposed method (*318 *sec.).*

Method	Time	Peak mem.	Triangles	Error
Power Crust	504	2601	1,610,433	4×10^{-4}
Hoppe *et al.*	82	230	630,345	6×10^{-4}
MPU	78	421	2,121,041	4×10^{-4}
FFT method	93	1700	1,458,356	6×10^{-4}
Poisson method	188	283	783,127	5×10^{-4}
Our method	75	236	1,292,609	4×10^{-4}

Table 2: *Computing time (seconds), peak-memory usage (mega-bytes), number of triangles and reconstruction error for the Stanford bunny by different methods (see also Fig. 6).*

does not cope well with the noise and non-uniformity of the data. Global methods such as the FFT, Poisson and our method accurately reconstruct the surface of the bunny, see Table 2. The smallest reconstruction error was achieved by the Power Crust, MPU and our method. Note that although the Power Crust method should have produced an interpolating surface, thus achieving a smaller upper-bound error, this does not happen in this case, as the reconstructed surface contains holes. The computing time of our method is comparable to that of the MPU method, which is one of the fastest reconstruction methods [OBA*03, KBH06]. Although the FFT method is also fast in this case, it becomes impractical at higher grid resolutions due to its large memory requirements [Kaz05]. The Poisson method is roughly two times slower than our method and has higher memory

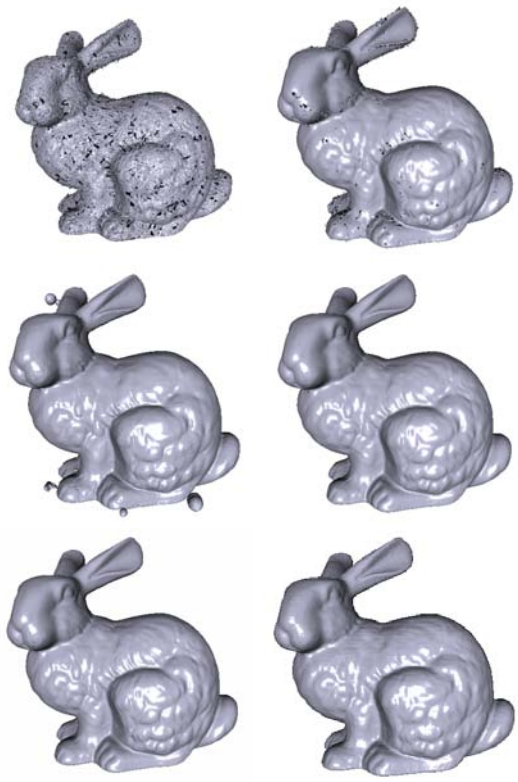

Figure 6: *Reconstruction of the Stanford bunny;* Left-to-right, top-to-bottom*: Power Crust, Hoppe* et al.*, MPU, FFT-based, Poisson reconstruction, and our method.*

usage. However, if larger accuracies are needed, geometrically adaptive methods such as the MPU and Poisson become more efficient.

Among the few methods which can tolerate a large amount of outliers is the recent one by Kolluri *et al.* [KSO04]. The CPU time reported in [KSO04] is one order of magnitude larger than that of our method, on the same input set (compare Fig. 1 in [KSO04] to Fig. 5). Additionally, we compared our method to the level-set method of Zhao *et al.* [ZOF01] also implemented on GPU hardware, and to that of Hoppe *et al.* [HDD*92], see Fig. 5. Our method is roughly twice as fast as Hoppe's method and the reconstructed surface is cleaner than that obtained by the level-set method which is much slower. Note that none of the other methods listed in Table 2 can cope with shot noise, and/or they require normal estimates at the sample points which in this case cannot be obtained.

5.4. Limitations

Surface features smaller than the grid resolution are not appropriately reconstructed. A solution is to increase the reso-

lution at the expense of larger computational time and memory requirements. The method is not geometrically adaptive, but we are currently investigating adaptive, multi-resolution approaches based on octrees. As is usual for methods that employ implicit surface representations, we assume that the surfaces to be reconstructed are closed, though the method does intrinsically perform hole filling by minimal surfaces.

6. Conclusions

We have shown that surface reconstruction can be formulated as a convection problem of a surface in a velocity field generated by Coulomb potentials, and that this formulation offers a number of advantages. The method can be used to efficiently reconstruct surfaces from clean as well as noisy and non-uniform real-world data sets. Further, it can deliver multi-resolution representations of the reconstructed surface, and can be used to perform reconstruction starting from particle systems, contours or even grey-scale volumetric data leading to image segmentation. In our implementation, we took advantage of the increased computational power of modern GPUs and ported most constituent parts of the method on graphics hardware.

References

[ACK01] AMENTA N., CHOI S., KOLLURI R.: The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications 19*, 2–3 (2001), 127–153. 6

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proc. SIGGRAPH'03* (2003), pp. 917–924. 4

[BLdL98] BECKERS J. V. L., LOWE C. P., DE LEEUW S. W.: An iterative PPPM method for simulating Coulombic systems on distributed memory parallel computers. *Mol. Sim. 20* (1998), 369–383. 3

[Blo94] BLOOMENTHAL J.: An implicit surface polygonizer. Academic Press Professional, Inc., San Diego, CA, USA, 1994, pp. 324–349. 1, 2, 3

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proc. SIGGRAPH'01* (2001), pp. 67–76. 2

[CBM*03] CARR J., BEATSON R., MCCALLUM B., FRIGHT W., MCLENNAN T., MITCHELL T.: Smooth surface reconstruction from noisy range data. In *Proc. Graphite 2003* (2003), pp. 119–126. 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *Proc. SIGGRAPH'96* (1996), pp. 303–312. 2

[DTS02] DINH H. Q., TURK G., SLABAUGH G.: Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Trans. Pattern Anal. Machine Intell.* (2002), 1358–1371. 2

[FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C.: Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3 (2005), 544–552. 2

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proc. SIGGRAPH'92* (1992), pp. 71–78. 2, 6, 7

[HE88] HOCKNEY R. W., EASTWOOD J. W.: *Computer simulation using particles*. Taylor & Francis, Inc., 1988. 1, 2, 3

[JR06] JALBA A. C., ROERDINK J. B. T. M.: Surface reconstruction from noisy data using regularized membrane potentials. In *Eurographics/IEEE VGTC Symposium on Visualization* (2006), pp. 83–90. 2, 4, 6

[Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. In *Eurographics Symposium on Geometry Processing* (2005), pp. 73–82. 2, 6, 7

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing* (2006), pp. 61–70. 2, 6, 7

[KSH04] KOJEKINE N., SAVCHENKO V., HAGIWARA I.: Surface reconstruction based on compactly supported radial basis functions. In *Geometric modeling: techniques, applications, systems and tools*. Kluwer Academic Publishers, 2004, pp. 218–231. 2

[KSO04] KOLLURI R., SHEWCHUK J. R., O'BRIEN J. F.: Spectral surface reconstruction from noisy point clouds. In *Symp. Geometry Processing* (2004), ACM Press, pp. 11–21. 7

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.: Multi-level partition of unity implicits. In *Proc. SIGGRAPH'03* (2003), pp. 463–470. 2, 6, 7

[OBS03] OHTAKE Y., BELYAEV A., SEIDEL H.: Multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Proc. Shape Modeling Int.* (2003), pp. 153–164. 2

[TM98] TANG C. K., MEDIONI G.: Inference of integrated surface, curve, and junction descriptions from sparse 3-D data. *IEEE Trans. Pattern Anal. Machine Intell. 20* (1998), 1206–1223. 2

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Trans. Graph. 21*, 4 (2002), 855–873. 2

[ZOF01] ZHAO H., OSHER S., FEDKIW R.: Fast surface reconstruction using the level set method. In *Proc. IEEE Workshop Variational and Level Set Methods in Computer Vision* (2001), pp. 194–202. 1, 2, 3, 7